

A0002. 变量与标准输入输出

上节课我们顺利搭建了 Perl 运行平台,并用 print 函数做了测试.这节课我们来讲讲变量与标准输入输出.

我们在计算,处理各类数据的时候,总会用到变量的.编程里面的变量[variable],跟数学里面的变量,大体意义是一样的.变量能够存储的内容,也同样是各种各样的.

编程中的标准输入[standard input, stdi],指的是从键盘输入数据(区别于从文件,从网络等),比如从键盘输入两个数来求和.标准输出[standard output, stdo]指的是将程序运行的结果,显示到显示器上(区别于存入文件中或通过网络发给他人),比如讲上述求和结果显示出来.标准输入输出[stdio]是我们最常用到的 IO 方式之一.当然,文件 IO 我们也很常用.文件我们在后面章节中再讲.

今天我们介绍的变量叫做标量.它是最常用的.其他类型,例如<数组>[array]或<哈希>[Hash],由于我们不是很迫切地需要它们,我会考虑在之后的章节讨论或不做深入讨论.标量可以从来存储整数,小数,或者字符串等.现在我们来通过例子讲解如何使用标量变量.

```
$x = 14;
```

这里,我们就定义了一个变量,叫做\$x,并且将其赋值成 14.

Perl 中的标量变量都是由(必须由)\$符开始,后面加一些字母数字下划线来组成变量名.变量命名有一些规则,这里不赘述(我也记不全=.=),只提示一些,详细的可以通过教材,网络查询<变量命名规则>.一般不是手特别贱也不太会出错.

{不要以数字开头(不要\$4y 这种);大小写是不同的字母,如\$da, \$Da, \$DA, \$dA 是四个不同的变量;最好不要跟系统的关键字同名吧,如\$sprint,虽然好像 perl 中可以这么做,但是这习惯就怪怪的=.=.}

变量是可以进行运算的,例如:

```
$a=15;
$b= -14.67;
$c= $a*$b;
$d=$a+200.6;
print $c;
print " ";
print $d;
```

这样,\$c 就是前两者的乘积了.\$d 同理.

程序中的赋值符等号=是这样操作的:先计算等号右边的式子的值,然后赋给左边的变量,于是我们可以对一个变量自身进行操作,例如:

```
$b = 5;
$x = 15.43;
$x = $x +1;    # x 自身加一,再存到 x 中,此时 x 变为 16.43
$x = $x + $b;  # 计算 x+b 的值,存入 x 中,此时 x 为 21.43,b 没变还是 5
print"x=$x\nb=$b\n";  ####!这句懂么?跑一下?是不是又更深理解 print 了?
```

数值变量的运算应该不是很难理解,如下是常用运算符:+(加),-(减),*(乘),/(除),**(乘幂),%(取余),-(单目负).其余更多知识可自行查阅<Perl 运算符>.

变量之中除了能够存数字,也能够存字符串,例如:

```
$s1="hello, perl!";  
$s2="I eat 5 oranges just now."  
$s3="gene_length";  
print "$s1\n$s2\n$s3\n";
```

这里,我们发现字符串都会用引号引起来.至此,我们使用的都是双引号,并且是那种不分左右的双引号.其实 perl 中也有用单引号的写法,比较繁琐,可以查阅书.我们就用双引号吧.

对于字符串,有很多操作,比如截成段[split],部分替换啊[replace],之类的,我们之后会专门就字符串操作展开讨论,今天就不讲了.

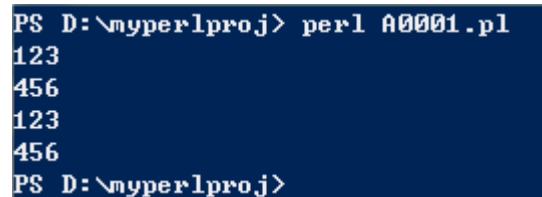
现在说标准输出.之前我们的用法是,print 加双引号.双引号中的\$变量名会被替换成变量的值,双引号中的转义字符会被替换成对应的字符,其他的字符基本都原样输出了.

要想实现标准输出,也可以用单引号,但是单引号和双引号用法略有不同,可以参考书.另外,除了 print 之外,也可以使用 printf 这个函数.printf 更加复杂,但是对于格式控制更好(比如输出几位小数,等).printf 的用法可以自己查一下,不展开了.

从键盘输入的方法很简单,我们来看这个例子(很有必要自己跑一下):

```
$s1=<>;  
$s2=<>;  
print "$s1$s2";
```

<>的作用就是从键盘获得一行内容,送给左边的变量\$s1.在键盘输入的方法是,输入一行字然后按回车.现在就让我们一起运行一下这个程序试试,我们第一行输入 123 送给 \$s1,第二行输入 456 送给 \$s2.运行结果如下:



```
PS D:\myperlproj> perl A0001.pl  
123  
456  
123  
456  
PS D:\myperlproj>
```

需要说明一下的是,当程序运行到<>这里时,屏幕才会进入等待输入的状态.如果由于某些原因没有执行<>这句,那么屏幕就不会进入等候输入的状态.如果<>这句之前有'别的语句',也肯定会依次序先运行'别的语句'.调用屏幕输入,只发生在运行到这句的时候.程序还是得按次序来的.请不要理解成:程序一开始就会让你输入东西,输入完毕,再从程序第一行开始运行.

第一组 123 456 是我们输入时留下的痕迹,后面的 123 456 是 print 出来的.仔细观察,有没有发现奇怪的地方?为什么第二个 123 后面换行了?第二个 456 后面也换行了!可是我们 print 的时候并没有换行啊!

这是为什么呢?其实呢,\$s1 和 \$s2 后面都是有一个换行符的,即 \$s1="123\n", \$s2="456\n".我键盘输入的时候命名只有 123,为什么多了个\n呢?这是因为,我们输入完毕之后会按一下回车,系统把我们按下的这个回车也作为输入内容赋值给左边的变量了.事实上,我们往往不想要末尾这个回车,这里有个办法能将其删去,那就是使用 chomp() 函数.

```
$s1=<>;
```

```

$s2=<>;
chomp($s1);
chomp($s2);
print("$s1$s2");
$s3=$s1+$s2;
print("\n$s3");

```

chomp 括号里放变量名,就能把变量末尾的换行符\n 去掉.再次运行,我们发现运行结果如下:

```

PS D:\myperlproj> perl A00001.pl
123
456
123456
579PS D:\myperlproj>

```

仔细看一下,对了,合我们心意啦!用<>获得输入之后,往往会依需要用 chomp() 处理一下,勉强算是好习惯吧.=.=

这里我们又学习到了一个新的函数 chomp().如果细心,可能有一个疑问,为什么 chomp 要用括号括起来他的操作对象(函数的操作对象我们称之为参数[parameter])?为什么之前 print 没有用括号?严谨地说,函数都应该写成这样子:函数名(参数们).之前我们没有给 print 写括号,是省略了.这种省略有一定方便性,但是也会造成困惑.于是我们之后使用函数的时候,包括 print,都给他加上括号好了.

有没有学会<>这个输入方式呢?现在我们把之前学的综合起来,出一个思考题.如果觉得有难度,没关系!

思考题:

我们来设计构思一个求两数和的小程序,会用到输入<>,chomp(),输出 print() .

程序开始的时候,显示一些欢迎信息,并且提示程序的功能.提示用户输入两个变量 x1,x2,然后输出它们的和.我来个结果截图:

程序开始:

```

PS D:\myperlproj> perl A00001.pl
*****
* This is a program to calculate the sum of two numbers *
* Pls input the value of these two numbers, x1, x2:      *
*****
x1=_

```

输入 x1 之后:

```

PS D:\myperlproj> perl A00001.pl
*****
* This is a program to calculate the sum of two numbers *
* Pls input the value of these two numbers, x1, x2:      *
*****
x1=678
x2=

```

输入 x2 之后出结果：

```
PS D:\myperlproj> perl A00001.pl
*****
* This is a program to calculate the sum of two numbers *
* Pls input the value of these two numbers, x1, x2:      *
*****
x1=678
x2=34
The sum of 678 and 34 is: 712
PS D:\myperlproj>
```

是不是很有爱呢？想想怎么实现这个功能。怎么实现前面有个变量名提示呢？怎么获得输入再求和呢？怎么输出呢？